

# CP 400S Project Report

## SQL Injection Attack

### Authors:

Jay Umesh Vora - Student ID: 203321900

Email Address - [vora1900@mylaurier.ca](mailto:vora1900@mylaurier.ca)

Yash Rojiwadia - Student ID: 203039360

Email Address - [roji9360@mylaurier.ca](mailto:roji9360@mylaurier.ca)

Course: CP 400S - Computer Security - Winter 2023

Date of Submission: [04/06/2023]

## Table of Contents

CP 400S Project Report .....	1
SQL Injection Attack .....	1
Introduction:.....	3
Background and Overview of SQL Injection Attacks:.....	3
How the SQL Injection Attacks works? .....	3
Types of SQL Injection Attacks .....	4
• In band attack .....	4
• Inferential attack .....	4
• Out of band attack:.....	4
Consequences of SQL Injection Attacks:.....	5
Real-world Examples of SQL Injection Attacks: .....	6
Our SQL Injection Attack Project:.....	7
Vulnerable Code:.....	7
Secure Code: .....	11
Preventing SQL Injection Attacks:.....	14
Conclusion: .....	15
References:.....	15

## **Introduction:**

SQL injection attacks have become one of the most prevalent and damaging threats to web applications in recent years. This report aims to provide an overview of SQL injection attacks, including their techniques, consequences, and countermeasures. We will also explore real-world examples of SQL injection attacks and their impact. The scope of this report is limited to SQL injection attacks on web applications and does not cover other types of attacks or security vulnerabilities.

This report gives understanding of SQL injection attacks and provides understanding of our project for preventing them.

## **Background and Overview of SQL Injection Attacks:**

SQL Injection Attacks are a type of cyber attack that target web applications by injecting malicious SQL (Structured Query Language) code into input fields, such as search bars, login forms, and other user inputs. This type of attack can have severe consequences, including data breaches, loss of confidentiality, financial losses, and damage to the reputation of the affected organization.

## **How the SQL Injection Attacks works?**

In order to execute a SQL injection attack, an attacker first identifies vulnerable web applications by exploiting common vulnerabilities, such as lack of input validation, improper error handling, and the use of dynamic SQL queries. Once a vulnerable application has been identified, the attacker injects SQL code into input fields. When the web application processes this input, it executes the injected SQL code, allowing the attacker to gain unauthorized access to sensitive information or modify the application's behavior.

## Types of SQL Injection Attacks

There are several types of SQL injection attacks:

### ❖ In band attack

- Tautology: In a tautology attack, the attacker injects SQL code that always evaluates to true. This can be used to bypass authentication mechanisms and gain access to sensitive information.
- End of line comment: In an End of Line Comment attack, the attacker injects a comment symbol at the end of a legitimate SQL query, followed by additional malicious code. The comment symbol tells the database to ignore everything that comes after it, allowing the attacker to inject their own code without disrupting the functionality of the original query.
- Piggybacked queries: In a Piggybacked Queries attack, the attacker injects multiple SQL queries into a single input field. This can allow the attacker to execute arbitrary SQL commands and gain access to sensitive data.

### ❖ Inferential attack

- Illegal/logically incorrect queries: In this type of attack, the attacker injects SQL code that is illegal or logically incorrect. This can cause the system to display error messages often revealing vulnerable or injectable parameters to an attacker.
- Blind SQL injection: In a Blind SQL Injection attack, the attacker does not receive any information from the database directly. Instead, the attacker uses boolean logic to infer whether a particular condition is true or false.

### ❖ Out of band attack:

In an Out-of-Band SQL Injection attack, the attacker uses a different communication channel to retrieve information from the database.

## Consequences of SQL Injection Attacks:

SQL injection attacks can have severe consequences, including:

- ❖ **Data Breaches:** Attackers can use SQL injection attacks to gain unauthorized access to sensitive data, such as usernames, passwords, credit card numbers, and other personally identifiable information. This can result in a data breach, which can be costly for organizations to recover from.
- ❖ **Loss of Confidentiality:** SQL injection attacks can compromise the confidentiality of sensitive data, which can harm individuals and organizations alike. For example, if an attacker gains access to medical records or financial information, it can have severe consequences for the affected individuals.
- ❖ **Financial Losses:** SQL injection attacks can also result in financial losses, such as theft of credit card information, which can be used to make unauthorized purchases. Organizations may also face financial penalties, such as fines or lawsuits, if they fail to protect user data.
- ❖ **Damage to Reputation:** A successful SQL injection attack can damage an organization's reputation and erode customer trust. This can result in a loss of business and revenue.

Overall, the consequences of a SQL injection attack can be severe and long-lasting. It is crucial for organizations to take proactive measures to prevent these attacks from occurring, such as implementing best practices and countermeasures, regularly monitoring for vulnerabilities, and educating their employees and users about the risks of cyber attacks.

## Real-world Examples of SQL Injection Attacks:

SQL injection attacks have been used to compromise the security of numerous high-profile organizations, including Yahoo!, Target, and Equifax.

- ❖ In 2013, hackers used SQL injection to breach the security of Yahoo! and gain access to the usernames and passwords of all three billion Yahoo! user accounts.
- ❖ The Target breach in 2013, which compromised the credit and debit card information of over 40 million customers, was also caused by a SQL injection attack.
- ❖ The Equifax breach in 2017, which exposed the personal information of 143 million individuals, was attributed to an unpatched vulnerability in a web application that was susceptible to SQL injection attacks.

These examples highlight the serious and widespread impact of SQL injection attacks, and the urgent need for organizations to take proactive measures to prevent such attacks from occurring.

## Our SQL Injection Attack Project:

### Vulnerable Code:

#### 1) Login (Username & Password) :

```
else
{
    $username = $_POST['username'];
    $password = $_POST['password'];

    $query = sprintf("SELECT * FROM users WHERE username = '%s' AND password = '%s';",
        $username,
        $password);

    $result = mysqli_query($conn, $query);

    if ($result->num_rows > 0)
    {
        echo "<p class='text-center'>Authenticated as <strong>" . $username . "</strong></p>";
        // ...
        // $_SESSION['logged_user'] = $username;
        // ...
    }
    else
    {
        echo "<p class='text-center'>Wrong username/password combination.</p>";
    }
}
?>
```

The above code is constructing a SQL query using string concatenation, which can be vulnerable to SQL injection attacks. An attacker can modify the input of the username or password fields to inject malicious SQL code that can bypass authentication and gain unauthorized access to the database.

#### SQL-Injection Demo

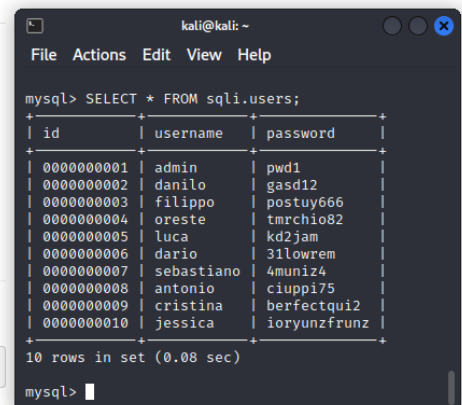
[Standard Login](#)[Numeric Login](#)[Search](#)[Tools](#)

#### Vulnerable Standard Login

| ID Number - 0000000001 | Username - admin | Password - pwd1 | ID Number - 0000000002 | Username - danilo | Password - gasd12 | ID Number - 0000000003 | Username - filippo | Password - postuy666 | ID Number - 0000000004 | Username - oreste | Password - tmrchio82 | ID Number - 0000000005 | Username - luca | Password - kd2jam | ID Number - 0000000006 | Username - dario | Password - 31lowrem | ID Number - 0000000007 | Username - sebastiano | Password - 4muniz4 | ID Number - 0000000008 | Username - antonio | Password - ciuppi75 | ID Number - 0000000009 | Username - cristina | Password - berfectqui2 | ID Number - 0000000010 | Username - jessica | Password - ioryunzfrunz

#### Query Executed:

```
SELECT * FROM users WHERE username = 'ABC' AND password = '1' OR '1'='1';
```



```
kali@kali: ~
File Actions Edit View Help

mysql> SELECT * FROM sqli.users;
+----+-----+-----+
| id  | username | password |
+----+-----+-----+
| 0000000001 | admin | pwd1 |
| 0000000002 | danilo | gasd12 |
| 0000000003 | filippo | postuy666 |
| 0000000004 | oreste | tmrchio82 |
| 0000000005 | luca | kd2jam |
| 0000000006 | dario | 31lowrem |
| 0000000007 | sebastiano | 4muniz4 |
| 0000000008 | antonio | ciuppi75 |
| 0000000009 | cristina | berfectqui2 |
| 0000000010 | jessica | ioryunzfrunz |
+----+-----+-----+
10 rows in set (0.08 sec)

mysql>
```

For example, an attacker can input the following in the username field: ' OR 1=1; --. This will modify the SQL query to SELECT \* FROM users WHERE username = ' OR 1=1; --' AND password = '<password>;', which will return all rows in the users' table, bypassing authentication.

## 2) Login (ID number & PIN):

```
else
{
    $client = $_POST['client'];
    $pin = $_POST['pin'];

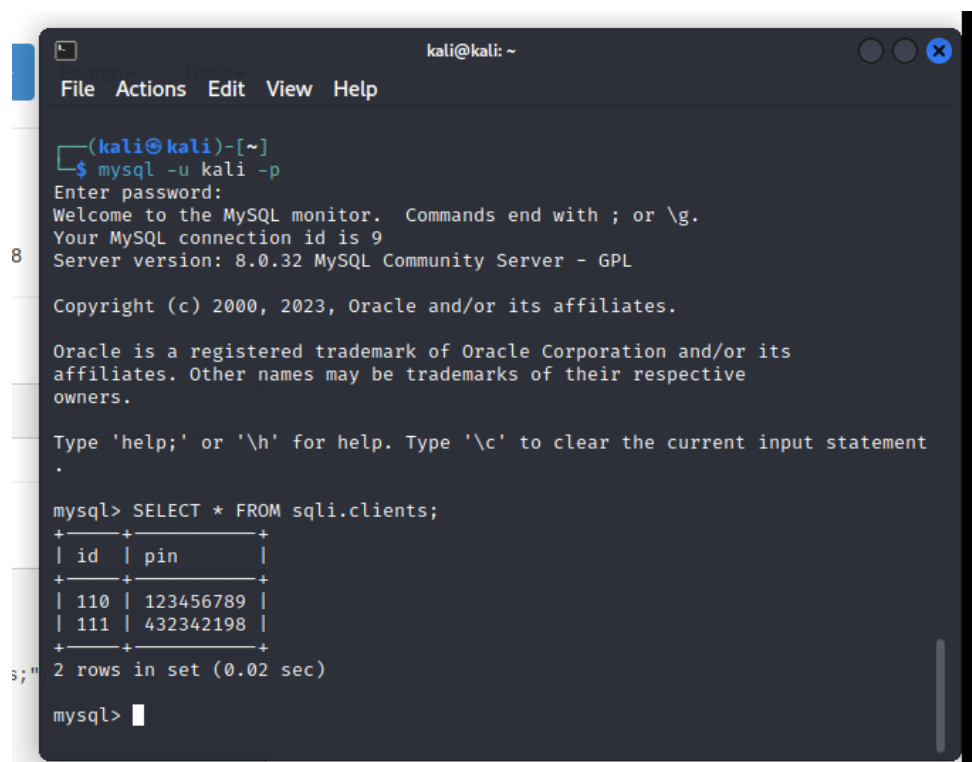
    $query = sprintf("SELECT * FROM clients WHERE id = %s AND pin = %s;",
        mysqli_real_escape_string($conn, $client),
        mysqli_real_escape_string($conn, $pin));

    $result = mysqli_query($conn, $query);

    if ($query != null)
    {
        $result = mysqli_query($conn, $query);

        while (($row = mysqli_fetch_row($result)) != null)
        {
            printf(" id number - <tr> <td>%s</td> | pin number - <td>%s</td> </tr> ", $row[0], $row[1]);
        }
    }
}
```

Here, the code is using string concatenation to construct a SQL query. An attacker can manipulate the input of the client or pin fields to inject malicious SQL code that can bypass authentication and gain unauthorized access to information from the database.



```
kali@kali: ~
File Actions Edit View Help

(kali@kali)-[~]
$ mysql -u kali -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 8.0.32 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement
.

mysql> SELECT * FROM sqli.clients;
+----+-----+
| id | pin  |
+----+-----+
| 110 | 123456789 |
| 111 | 432342198 |
+----+-----+
2 rows in set (0.02 sec)

mysql>
```



**Vulnerable Numeric Login**

| id number - 110 | pin number - 123456789 | id number - 111 | pin number - 432342198

Query Executed:

```
SELECT * FROM clients WHERE id = 12 AND pin = 1 OR 1=1;
```

As it can be seen after exploiting the vulnerabilities, we can extract the id numbers and pin numbers from the database using SQL queries in the input box.

### 3) Search (Books):

```
<?php
if ($_GET['all'] == 1)
{
    $query = "SELECT * FROM books;";
}
else if ($_GET['title'] || $_GET['author'])
{
    $query = sprintf("SELECT * FROM books WHERE title = '%s' OR author = '%s';",
                    $_GET['title'],
                    $_GET['author']);
}

if ($query != null)
{
    $result = mysqli_query($conn, $query);

    while (($row = mysqli_fetch_row($result)) != null)
    {
        printf("<tr><td>%s</td><td>%s</td><td>%s</td></tr>", $row[0], $row[1], $row[2]);
    }
}
?>
</table>
```

The code is using string concatenation to build a SQL query for the title and author parameters. An attacker can potentially manipulate the input of these parameters to inject malicious SQL code, leading to SQL injection vulnerabilities.

The code is using printf() to output the query results without proper input sanitization. This can allow an attacker to inject malicious scripts or HTML code into the output, leading to XSS vulnerabilities.

SQL-Injection Demo    Standard Login ▾    Numeric Login ▾    Search ▾    Tools ▾

**Vulnerable Search**

Book title  Book author  Search

#ID	Title	Author
1	admin	pwd1
2	danilo	gasd12
3	filippo	postuy666
4	oreste	tmrchio82
5	luca	kd2jam
6	dario	31lowrem
7	sebastiano	4muniz4
8	antonio	ciuppi75
9	cristina	berfectqui2
10	jessica	ioryunzfrunz

```
kali@kali: ~  
File Actions Edit View Help  
mysql> SELECT * FROM sql.users;  
+----+-----+-----+  
| id | username | password |  
+----+-----+-----+  
| 000000001 | admin | pwd1 |  
| 000000002 | danilo | gasd12 |  
| 000000003 | filippo | postuy666 |  
| 000000004 | oreste | tmrchio82 |  
| 000000005 | luca | kd2jam |  
| 000000006 | dario | 31lowrem |  
| 000000007 | sebastiano | 4muniz4 |  
| 000000008 | antonio | ciuppi75 |  
| 000000009 | cristina | berfectqui2 |  
| 000000010 | jessica | ioryunzfrunz |  
+----+-----+-----+  
10 rows in set (0.08 sec)  
mysql>
```

As can be seen from the above example that after executing a simple query such as " ' UNION SELECT \* FROM users WHERE '1'='1" into the input box, the attacker can extract all the users' information such as their usernames and IDs.

## Secure Code:

### 1) Login (Username & Password):

```
else
{
    $username = $_POST['username'];
    $password = $_POST['password'];

    $query = sprintf("SELECT * FROM users WHERE username = '%s' AND password = '%s';",
        mysqli_real_escape_string($conn, $username),
        mysqli_real_escape_string($conn, $password));

    $result = mysqli_query($conn, $query);

    if ($result->num_rows > 0)
    {
        echo "<p class='text-center'>Authenticated as <strong>" . $username . "</strong></p>";

        // ...
        // $_SESSION['logged_user'] = $username;
        // ...
    }
    else
    {
        echo "<p class='text-center'>Wrong username/password combination.</p>";
    }
}
```

The code is using Encoding to encode the user input to build a SQL query for the username or password fields. This way if attacker tries to insert any SQL code, it will be considered as data instead of code due to already given ( ' ' ) around the username and password preventing any code to execute preventing unauthorized access to the database.

## SQL-Injection Demo

[Standard Login ▾](#)[Numeric Login ▾](#)[Search ▾](#)[Tools ▾](#)[Secure Standard Login](#)

Wrong username/password combination.

### Query Executed:

```
SELECT * FROM users WHERE username = 'ABC\' OR 1=1--' AND password = '';
```

As it can be seen after trying SQL injection, we can not extract the usernames or passwords from the database using SQL queries in the input box due to input converted into data instead of code.

## 2) Login (ID number & PIN):

```
else
{
    $client = $_POST['client'];
    $pin = $_POST['pin'];

    if (is_numeric($client) && is_numeric($pin))
    {
        $query = sprintf("SELECT * FROM clients WHERE id = %s AND pin = %s;",
                        mysqli_real_escape_string($conn, $client),
                        mysqli_real_escape_string($conn, $pin));

        $result = mysqli_query($conn, $query);

        if ($result->num_rows > 0)
        {
            echo "<p class='text-center'>Authenticated as <strong>" . $client . "</strong></p>";

            // ...
            // $_SESSION['logged_user'] = $client;
            // ...
        }
        else
        {
            echo "<p class='text-center'>Wrong client/PIN combination.</p>";
        }
    }
    else
    {
        echo "<p class='text-center'>Client ID and PIN must be numeric values.</p>";
    }
}
```

The code is using Encoding to encode the user input to build a SQL query for the id number and pin number fields. This way if attacker tries to insert any SQL code, it will be considered as data instead of code due to specific data type defined which is numeric and already given ( ' ') around the id number and pin number preventing any code to execute preventing unauthorized access to the database.

### SQL-Injection Demo

Standard Login▼

Numeric Login▼

Search▼

Tools▼

Secure Numeric Login

Client ID and PIN must be numeric values.

Query Executed:

As it can be seen after trying SQL injection, we can not extract the id numbers and pin numbers from the database using SQL queries in the input box due to input not being numerical value.

### 3) Search (Books):

```
if ($_GET['all'] == 1)
{
    $query = "SELECT * FROM books;";
}
else if ($_GET['title'] || $_GET['author'])
{
    $query = "SELECT * FROM books WHERE title = ? OR author = ?";
    $stmt = mysqli_prepare($connection, $query);
    mysqli_stmt_bind_param($stmt, "ss", $_GET['title'], $_GET['author']);
}

if ($query != null)
{
    $result = mysqli_stmt_execute($stmt);
    mysqli_stmt_bind_result($stmt, $col1, $col2, $col3);

    while (mysqli_stmt_fetch($stmt))
    {
        printf("<tr><td>%s</td><td>%s</td><td>%s</td></tr>", $col1, $col2, $col3);
    }
}
```

The code is using Parameterized user input to build a SQL query the title and author fields. This way if attacker tries to insert any SQL code, it will be separating user input from the SQL code preventing unauthorized access to the database.

#### SQL-Injection Demo

[Standard Login](#)[Numeric Login](#)[Search](#)[Tools](#)[Secure Search](#)

#ID

Title

Author

Query Executed:

```
FROM books WHERE title = 'ABC' OR author = '' UNION SELECT * FROM users WHERE \'1\'=\'1';
```

As it can be seen after trying SQL injection, we can not extract the user data from the database using SQL queries in the input box due to inputs being separated into user input and the SQL code.

## Preventing SQL Injection Attacks:

Preventing SQL injection attacks is crucial for ensuring the security of organizations and their users. Here are some techniques that can be used to prevent SQL injection attacks:

1. **Input Validation:** One of the most effective ways to prevent SQL injection attacks is to ensure that all user input is validated before it is processed by the application. This includes checking for the correct data type, length, format, and range of input values. By validating input, organizations can prevent attackers from injecting malicious SQL code into the application.
2. **Parameterized Queries:** Parameterized queries can help prevent SQL injection attacks by separating SQL code from user input. Parameterized queries use placeholders for user input values, which are then replaced with the correct value before the query is executed. This approach makes it more difficult for attackers to inject malicious code into the application.
3. **Stored Procedures:** Stored procedures can also help prevent SQL injection attacks by encapsulating SQL code and making it more difficult for attackers to manipulate. Stored procedures are precompiled and can be reused across multiple queries, reducing the risk of SQL injection attacks.
4. **Encoding (turning code into data):** Encoding is a method of transforming data into a different format to prevent malicious code injection. Encoding can be used to prevent SQL injection attacks by encoding user input before it is included in an SQL query. This makes it more difficult for an attacker to inject malicious code into the application.

By implementing these prevention techniques, organizations can significantly reduce their risk of SQL injection attacks. Additionally, educating employees and users about the risks of SQL injection attacks and best practices for preventing them can also help improve the overall security posture of organizations.

## Conclusion:

In conclusion, SQL injection attacks are a serious threat to the security of organizations and their users. These attacks can have severe consequences, including data breaches, loss of confidentiality, financial losses, damage to reputation, and legal consequences. It is crucial for organizations to take proactive measures to prevent SQL injection attacks, such as implementing best practices and countermeasures, regularly monitoring for vulnerabilities, and educating their employees and users about the risks of cyber attacks. By implementing these prevention techniques, organizations can significantly reduce their risk of SQL injection attacks and improve their overall security posture. It is essential to remain vigilant and stay up-to-date with the latest security trends and technologies to protect against this pervasive and ever-evolving threat.

## References:

Gao, S. (2023). CP400S-Computer Security-week2-1-Web-Security. CP-400S Computer Security.

Gao, S. (2023). CP400S-Computer Security-week2-2-SQL-Injection. CP-400S Computer Security.

Borzi, F. (2014). Francescoborzi/SQL-Injection-DEMO: Computer security project. GitHub. Retrieved April 6, 2023, from <https://github.com/FrancescoBorzi/sql-injection-demo>